

一种基于二次移动平均法的容器云伸缩策略

刘钱超,吴利,郑礼辉

(江南计算技术研究所,江苏无锡 214083)

摘要:容器云作为一种高效、易维护和低成本的云环境解决方案,弥补了传统基于虚拟机的云环境构建方式的不足,受到国内外软件服务提供商的广泛关注。针对容器云基于阈值的响应式伸缩策略存在时间滞后性、难以及时响应服务资源请求的问题,提出一种基于二次移动平均法的预测式容器云伸缩方法。该方法在检测实时工作负载的同时通过二次移动平均法对未来工作负载进行预测,然后基于实时负载值和预测负载值利用响应式伸缩策略进行容器云伸缩决策。与原来基于阈值的响应式伸缩策略相比,在 Docker swarm 集群环境并施加周期性负载的实验条件下,提出的优化方法能够提前预测负载变化并调整集群规模,有效应对负载波动,服务响应时间波动幅度降低了约 42.9%,保证了容器云中应用服务的质量和稳定性。

关键词:容器云;伸缩策略;虚拟机;Docker swarm;应用服务质量

中图分类号:TP301

文献标识码:A

文章编号:1673-629X(2019)10-0015-06

doi:10.3969/j.issn.1673-629X.2019.10.004

A Container Cloud Scaling Strategy Based on Double Moving Average Method

LIU Qian-chao, WU Li, ZHENG Li-hui

(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

Abstract: As an efficient, easy-to-maintain and low-cost cloud environment solution, container cloud makes up for the shortcomings of traditional virtual machine-based cloud environment construction, which has been widely concerned by software service providers at home and abroad. Aiming at the problem that the threshold-based responsive scaling strategy of container cloud has time lag and difficulty in responding to service resource request in time, we propose a predictive container cloud scaling method based on double moving average method. The method predicts the future workload by means of the double moving average method while detecting the real-time workload, and then uses the responsive scaling strategy to make the container cloud scaling decision based on the real-time workload value and the predicted workload value. Compared with the original threshold-based responsive scaling strategy, under the conditions of Docker swarm cluster environment and periodic workload, the proposed optimization method can predict the workload change in advance and adjust the cluster scale to deal with the workload fluctuation effectively. The fluctuation range of service response time is reduced by 42.9%, which ensures the quality and stability of service in container cloud.

Key words: container cloud; scaling strategy; virtual machine; Docker swarm; application quality of service

0 引言

随着大数据、云计算等软件技术的飞速发展,基于虚拟机集群的分布式软件运行环境构建方式资源开销大、耗时长、可伸缩性差等缺点逐渐显露,降低了软件运行效率,增加了运维成本,而具有高可伸缩性和高资源使用率等优势的容器云环境为上述问题提供了优化方案^[1-2]。

容器云是以容器为资源分割和调度的基本单位,

封装整个软件运行时环境,为开发者和系统管理者提供用于构建、发布和运行分布式应用的平台^[3]。基于容器云构建软件运行环境,与基于虚拟机集群方式相比,具有简化软件部署方式、优化管理运维模式和降低运维成本等优势。然而,容器云作为一种全新的云环境解决方案,在集群动态伸缩策略方面仍存在不足。集群的动态伸缩^[4-6]是指集群能够根据工作负载的变化动态地调整自身规模,从而调整其对外服务的能力。

收稿日期:2018-11-19

修回日期:2019-02-19

网络出版时间:2019-04-24

基金项目:国家重点研发计划(2016YFB1000505)

作者简介:刘钱超(1994-),男,硕士研究生,研究方向为软件测试、大数据、云计算。

网络出版地址: <http://kns.cnki.net/kcms/detail/61.1450.TP.20190424.1100.084.html>

目前,容器云采用的是基于阈值的响应式伸缩策略,该策略只会对系统资源的实时情况进行响应,因而系统的伸缩调整往往会晚于工作负载的变化,导致服务质量不佳,用户体验较差。

目前,已有很多关于云环境下集群动态伸缩策略的改进研究。文献[7]提出一种基于 workflow 模型的虚拟机动态分配和回收策略,能够较大程度地提高系统资源利用率,但该策略不能对集群未来工作负载进行预测。文献[8]针对基于虚拟机的资源管理难以实现细粒度的资源动态调整与混合云中跨平台服务快速迁移的问题,提出一种基于容器技术的云计算资源自适应管理方法,能够提高集群自适应响应速度和集群整体性能,但该方法也不能对集群未来工作负载进行预测。文献[9]提出一种基于集群负载预测的虚拟机资源管理框架,能够实现对未来工作负载预测并及时做出伸缩响应,保证服务质量,但该框架底层是基于虚拟机资源实现,由于虚拟机启停耗时长,集群伸缩响应速度仍然较慢。

针对文献[7-8]不能对未来工作负载进行预测和文献[9]中基于虚拟机资源响应速度慢的不足,提出一种基于二次移动平均法的预测式容器云伸缩方法。该方法可以对未来工作负载进行预测并依托容器轻量特点快速做出伸缩响应,从而降低服务响应时间的波动幅度,提高服务的质量和稳定性。

1 伸缩策略选择与预测模型建立

1.1 集群动态伸缩技术

集群动态伸缩技术包括响应式伸缩^[10]和预测式伸缩^[11]两种。

(1) 响应式伸缩。

基于阈值的响应式伸缩策略因算法实现简单,对集群突发式负载具有较好效果,目前被广泛应用于云平台伸缩决策中。响应式伸缩策略需要对响应规则和阈值进行设定。算法运行时,会对系统内资源(CPU、网络、存储、内存等)进行监控,一旦监测到某些指标超过既定阈值,集群会发出相应的扩展或收缩命令。具体来说,如图1(a)所示, t_1 时刻通过对系统资源进行持续监控,当 t_2 时刻监测到系统资源有较大变化且超过阈值(包括上下限阈值)时,开始进行集群伸缩, t_5 时刻完成伸缩。基于阈值的响应式伸缩策略存在的不足是无法对未来负载进行预测,只能对系统资源的实时情况进行响应,因而系统的伸缩调整往往会晚于工作负载的变化,导致服务质量不佳,用户体验变差。

(2) 预测式伸缩。

预测式伸缩策略是对过去一段时间内系统资源使用情况进行分析,然后利用数学方法进行建模并完成

对未来时刻的预测。它能够较好克服响应式伸缩策略存在时间滞后性的缺点。具体来说,如图1(b)所示,当 t_1 时刻根据历史数据特点预测到 t_6 时刻系统资源使用发生较大改变且超过所设置的阈值时,系统会在 t_2 时刻发出伸缩命令,因而系统能在 t_6 时刻负载发生大幅变化之前有充足时间完成伸缩操作,从而能够及时响应系统资源变化。

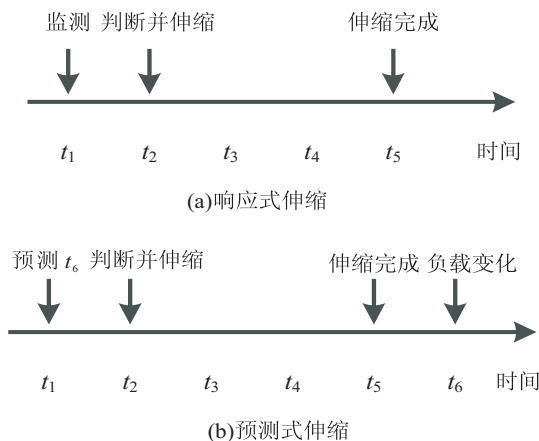


图1 两种伸缩方式

根据上述对两种伸缩策略的分析,文中结合使用预测式伸缩策略与响应式伸缩策略,在对集群突发式负载具有良好响应效果的同时,也可以实现对未来时刻负载值的预测,克服响应式伸缩策略存在时间滞后性的不足,从而提高集群资源响应速度,更好地保障应用服务的质量和稳定性。

1.2 预测模型建立

预测式伸缩策略常采用的分析方法主要有时间序列分析和机器学习^[12],但机器学习需要较长模型训练时间,适用性不强。

二次移动平均法(又名二重移动平均法)是一种时间序列分析方法,它是在预测目标时间序列上的一次移动平均值的基础上再进行一次移动平均,并将结果代入到相关公式中得到预测值,目的是解决一次移动平均法存在滞后偏差,使得预测值总是落后于实际观测值的问题。它适用于数据随某个变量呈现自相关特性的场景,考虑到容器云中应用的资源使用情况可能随时间呈现一定规律,因此文中选取二次移动平均法对未来时刻负载值进行预测。其数学计算公式如下:

假设 $x_t, x_{t-1}, \dots, x_{t-n+1}$ 为系统在时刻 t 的历史数据序列,因此 t 时刻数据序列的一次移动平均值为:

$$M_t^{(1)} = \frac{1}{n}(x_t + x_{t-1} + \dots + x_{t-n+1}) \quad (1)$$

t 时刻一次移动平均值序列为: $M_t^{(1)}, M_{t-1}^{(1)}, \dots, M_{t-n+1}^{(1)}$ 。

t 时刻的二次移动平均值为:

$$M_t^{(2)} = \frac{1}{n}(M_t^{(1)} + M_{t-1}^{(1)} + \cdots + M_{t-n+1}^{(1)}) \quad (2)$$

$t + T$ 时刻预测值 x_{t+T}^{Pre} 为:

$$x_{t+T}^{\text{pre}} = a_t + b_t \cdot T \quad (3)$$

$$a_t = 2M_t^{(1)} - M_t^{(2)} \quad (4)$$

$$b_t = \frac{2}{n-1}(M_t^{(1)} - M_t^{(2)}) \quad (5)$$

因此基于二次移动平均法,对 t 时刻历史数据序列进行分析,可以计算出 $t + T$ 时刻的预测值:

$$x_{t+T}^{\text{pre}} = 2M_t^{(1)} - M_t^{(2)} + \frac{2}{n-1}(M_t^{(1)} - M_t^{(2)}) \cdot T \quad (6)$$

2 容器云动态伸缩方案

2.1 容器云伸缩决策过程

图 2 为文中提出的容器云伸缩算法的决策过程。在算法运行过程中,系统内会维护一张容器服务最近 L 次响应时间记录表 H ,通过对系统内容器服务 (Service_ID) 的响应时间 t^{res} 进行持续监控并用获取到的数据不断更新记录表 H 。决策模块会对表 H 中最近 L 次响应时间进行处理、建模、预测,得到未来时刻的预测值 t^{pre} ,根据实时响应时间数据以及预测值,利用基于阈值的响应式伸缩策略进行系统伸缩决策,并将伸缩命令传递给伸缩执行模块执行。执行完毕后,判断伸缩算法是否继续运行,根据判断结果来决定继续循环或终止该算法。

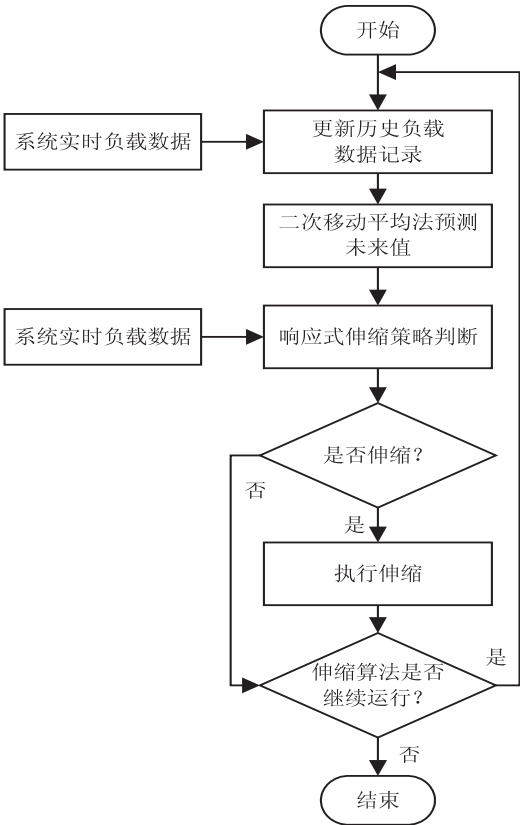


图 2 容器云伸缩决策过程

2.2 伸缩算法模块设计

如图 3 所示,伸缩算法包含 Monitor、Processor、Commander 和 Scheduler 四个模块。

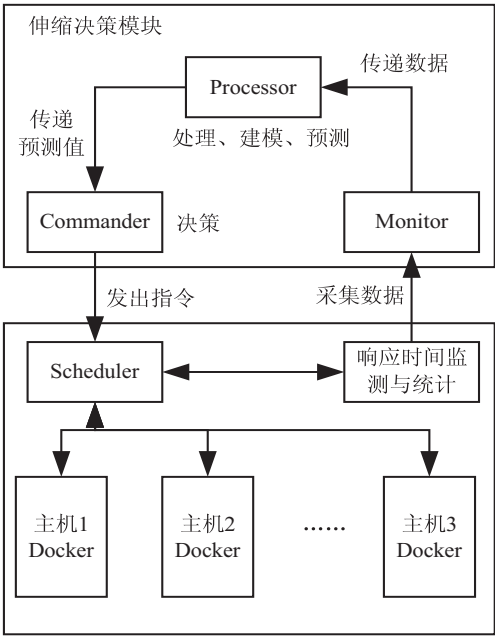


图 3 伸缩算法模块

(1) Monitor 模块。

Monitor 模块负责对应用服务响应时间 t^{res} 持续监控,并用获取到的响应时间序列不断更新表 H ,使表 H 始终保持存储最近 L 次应用服务响应时间。设定更新表 H 的策略为每次获取服务响应时间 t^{res} 后,将其加入到链表尾部,同时将链表首元素删除。在 Monitor 获取到服务响应时间后,同时也将采集到的数据传递给 Commander 模块。

(2) Processor 模块。

Processor 模块主要对表 H 中最近 L 次服务响应时间数据进行分析、建模和对未来时刻值的预测。在具体算法实现时,假设表 H 中存储的最近 L 次服务响应时间序列为: $x_{t-L+1}, x_{t-L+2}, \cdots, x_{t-1}, x_t$, 设置好二次移动平均法参数 n 和预测间隔 T 后,基于二次移动平均法,根据表 H 中 L 条数据使用式 6 预测 $t + T$ 时刻的服务响应时间值 x_{t+T} ,并将预测值传递给 Commander 模块。

(3) Commander 模块。

Commander 模块会对 Monitor 模块采集到的实时服务响应时间 t^{res} 和 Processor 模块利用二次移动平均法计算出的响应时间预测值 t^{pre} 依据基于阈值的响应式伸缩策略进行判断,如果 $t^{\text{res}} > T^{\text{u}} \parallel t^{\text{pre}} > T^{\text{u}}$ (T^{u} 为响应时间阈值上限) 关系成立,说明系统内服务响应时间过长,质量较差,因此 Commander 模块会向 Scheduler 模块发出扩展命令,由 Scheduler 模块完成容器服务扩展(算法 1 中 4~6 行)。如果 $t^{\text{res}} < T^{\text{l}} \parallel t^{\text{pre}} <$

T^d (T^d 为响应时间阈值下限) 关系成立, 说明系统内部服务质量处于优良状态, 因此为了提高系统资源利用率, 节约硬件运维成本, Commander 模块会向 Scheduler 模块发出收缩命令, 由 Scheduler 模块完成停止容器服务和资源回收工作(算法 1 中 7~9 行)。

算法 1: 伸缩决策算法。

```
Input:  $t^{res}$ ,  $H$ ,  $T^u$ ,  $T^d$ , Service_ID
Output: Scale_Dec
1 Begin
2 While( algorithm is running)
3 Monitor and Collect  $t^{res}$ , Maintain the table  $H$ , calculate the value  $t^{pre}$ 
4 if  $t^{res} > T^u$  ||  $t^{pre} > T^u$ 
5 send( Extension,  $E$ , Service_ID ) command to Scheduler
6 end if
7 if  $t^{res} < T^d$  ||  $t^{pre} < T^d$ 
8 send( Contraction,  $E$ , Service_ID ) command to Scheduler
9 end if
10 end While
11 End
```

(4) Scheduler 模块。

Scheduler 模块保存了一份对所有主机的评分表。集群中所有备选主机的 CPU 使用率、内存占用率以及网络带宽占用率每隔一段时间就会被 Scheduler 模块收集, 并依据式 7 进行评分和记录到评分表中, 通过评分高低来决定在哪台主机上进行容器服务的伸缩。

$$Score_i = (3 - CPU_use - Mem_use - Net_use),$$
$$Score_i \in (0, 3) \tag{7}$$

其中, CPU_use 为 CPU 使用率; Mem_use 为内存占用率; Net_use 为网络带宽占用率。

同时, 扩展时需要计算扩展容器服务个数 N 。假设需要扩展的资源为 E , 单个容器持有资源为 E_0 , 因此可得 $N = E/E_0$ 。当需要进行收缩资源时, 为防止因系统突然减少大量资源造成系统运行异常, 设定收缩资源时, 每次释放 1 个容器服务。

Scheduler 模块在运行过程中, 需要接收由 Commander 模块发出的命令, 如果收到的是扩展命令, 选择当前打分表中分数最高的主机, 根据服务 ID 及新建容器命令, 在指定主机上部署 N 个指定 ID 的容器, 对外提供服务并将其记录到 Service 记录表 S 中, 扩展工作结束。此时为了避免因频繁伸缩导致系统剧

烈“抖动”, 伸缩算法进入一小段时间的休眠期(算法 2 中 3~8 行); 如果收到的是收缩命令, 选择当前打分表中分数最低且其上运行着相关服务的主机。分数低意味着其上负载较重, 因此在释放资源时, 为了维持负载均衡, 首先移除负载较重主机上的服务, 减轻其上负载压力。主机选定后, 根据容器服务的 ID, 使用相关命令移除 1 个容器服务, 回收资源并更新 Service 记录表 S 的内容, 收缩工作结束。同样, 为了避免因频繁伸缩导致系统剧烈“抖动”, 伸缩算法进入一小段时间的休眠期(算法 2 中 9~14 行)。

算法 2: 伸缩执行算法。

```
Input: ( COMMAND,  $E$ , Service_ID )
Output: Updated S
1 Begin
2 While( algorithm is running)
3 if “COMMAND” equals “Extension”
4 calculate  $N = E/E_0$ ;
5 Select the highest Score of the physical machine and deploy  $N$ 
6 Service_ID container service, update  $S$ ;
7 Sleep( a period of time );
8 end if
9 if “COMMAND” equals “Contraction”
10  $N = 1$ ;
11 Select the lowest Score of the physical machine and remove  $N$ 
12 Service_ID container service, update  $S$ ;
13 Sleep( a period of time );
14 end if
15 end While
16 End
```

3 实验验证及结果分析

3.1 实验环境

实验搭建了两个四节点的 Docker swarm^[13] 集群, 分别为集群 1 和集群 2。集群 1 为无任何修改的 Docker swarm 集群, 即原生集群, 采用的是响应式伸缩策略。集群 2 为依据文中提出的伸缩策略改进后的 Docker swarm 集群。实验需要使用的工具包括并发测试执行工具 Jmeter 以及反向代理器 Nginx。实验硬件包含 6 台 PC 机, 各节点配置及安装环境如表 1 所示。

表 1 集群信息

主机名	软件环境	IP	硬件配置
Master	swarm 主节点	192.168.1.100	Intel Core2 Duo CPU @ 3.00 GHz × 2; 1.7 GB; 20 G
Node1	swarm 从节点 1	192.168.1.101	Intel Core2 Duo CPU @ 3.00 GHz × 2; 1.7 GB; 20 G
Node2	swarm 从节点 2	192.168.1.102	Intel Core2 Duo CPU @ 3.00 GHz × 2; 1.7 GB; 20 G
Node3	swarm 从节点 3	192.168.1.103	Intel Core2 Duo CPU @ 3.00 GHz × 2; 1.7 GB; 20 G

续表 1

主机名	软件环境	IP	硬件配置
Nginx	安装 Nginx 服务	192.168.1.104	Intel Core2 Duo CPU @ 3.00 GHz × 2; 1.7 GB; 20 G
JMeter	安装 JMeter 工具	192.168.1.105	Intel Core2 Duo CPU @ 3.00 GHz × 2; 1.7 GB; 20 G

图 4 为实验过程中主机之间的访问顺序。测试机通过其上安装的 Jmeter 测试软件以并发方式访问 swarm 集群中容器 Web 认证服务(实验条件下只生成了一类服务),并记录服务响应时间的变化情况。同时为了更好地验证提出的伸缩方法,在访问链路中加入了 Nginx 服务器来进行消息转发。

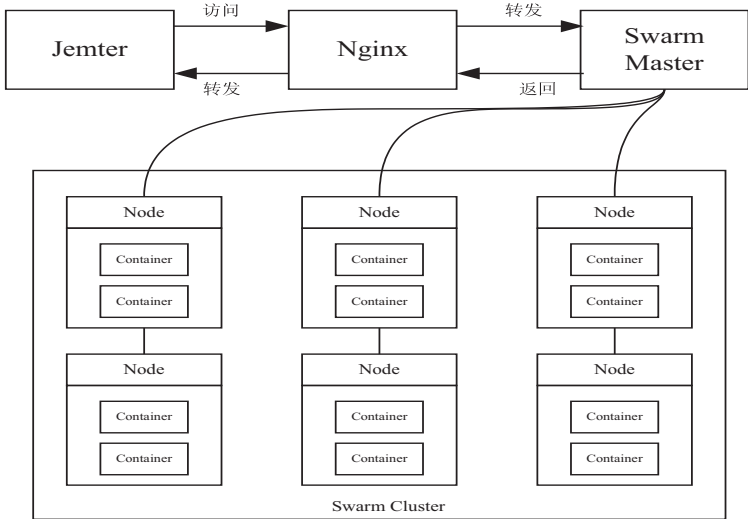


图 4 实验过程中主机之间的访问顺序

文中提出的伸缩策略在具体算法实现时, L 值设置为 10,容器扩展和收缩个数 N 设置为 1,响应时间阈值上限设置为 0.9 s,阈值下限设置为 0.2 s。二次移动平均法中 n 设置为 2,预测时刻间隔 T 设置为 1,也即根据 t 时刻的 3 个响应时间值,使用二次移动平均法可以预测 $t + 1$ 时刻的响应时间值。

3.2 实验测试数据

表 2 为实验负载数据。实验负载数据分为两类:周期性负载数据和突发式负载数据。

表 2 实验负载数据

轮次	周期性负载数据(请求数)	突发式负载数据(请求数)
1	50	50
2	120	120
3	230	70
4	320	380
5	410	110
6	500	130
7	390	170
8	280	430
9	120	230
10	30	350
11	110	40
12	210	310
13	320	130

续表 2

轮次	周期性负载数据(请求数)	突发式负载数据(请求数)
14	450	230
15	510	180
16	600	270

3.3 实验结果及分析

图 5 为施加周期性负载的实验结果。

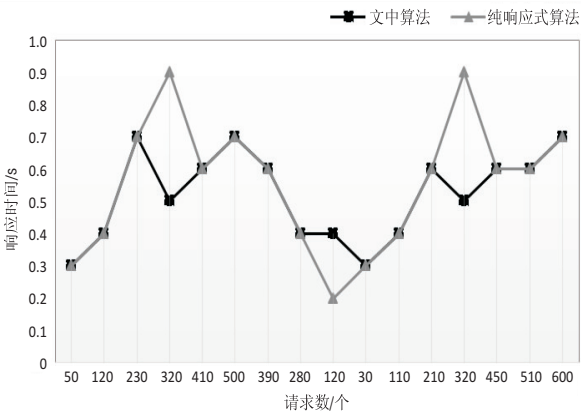


图 5 周期性负载实验结果

当请求数为 50、120 时,文中提出的算法和纯响应式伸缩算法效果一致,两条曲线在图中完全重合。原因是此时服务的实时响应时间和响应时间的预测值均未超过响应时间的阈值上限,没有触发容器云集群的伸缩机制。当请求数增长至 230 时,文中算法下个时

刻的服务响应时间预测值已经超出了阈值上限,因此会选择一台分数最高的主机新建一个对应容器并提供服务。当请求数继续增长至 320 时,由于文中算法已提前扩展了容器服务,服务的响应时间减至约为纯响应式伸缩算法的一半,而此时纯响应式伸缩算法才判定响应时间超过阈值上限并开始扩展。直到请求数为 410 时,纯响应式算法扩展效果开始显现,与文中算法效果一致。收缩过程与上述扩展过程类似。

从图 5 可以看出,对纯响应式伸缩策略,容器服务响应时间波峰为 0.9 s,波谷为 0.2 s,波动幅度为 0.7;而文中提出的优化算法服务响应时间波峰为 0.7 s,波谷为 0.3 s,波动幅度为 0.4。与纯响应式伸缩策略相比,文中算法的服务响应时间的波动幅度降低了约 42.9%,有效减弱了由负载变化引起的服务响应时间波动。

图 6 为施加突发式负载的实验结果。

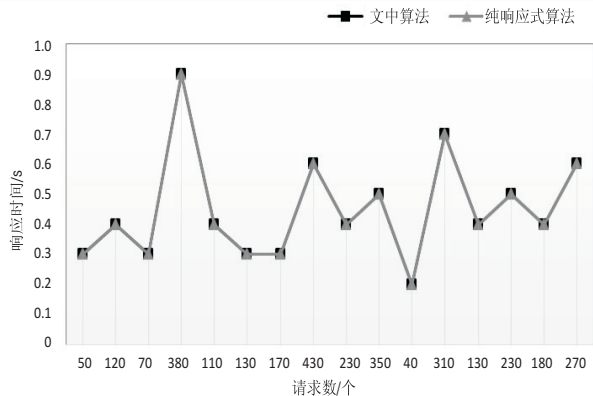


图 6 突发式负载实验结果

从图中可以看出,文中算法与纯响应式伸缩算法结果始终保持一致,两条曲线完全重合。原因为对改进伸缩策略后的 Docker swarm 集群施加突发式负载时,二次移动平均法无法对未来时刻值进行准确预测,导致预测效果丧失,集群伸缩决策只由服务的实时响应时间和响应时间阈值之间大小关系决定。因此,对于突发式负载,文中算法与纯响应式伸缩算法表现一致。

3.4 实验结论

文中提出的基于二次移动平均法的预测式容器云伸缩策略,在保留基于阈值的响应式伸缩策略对突发式负载具有良好响应效果的同时,在周期性负载条件下还能对未来负载值进行准确预测,提高了容器云对应用资源请求的响应能力,降低了服务响应时间波动幅度,保证了服务响应时间的稳定性。

4 结束语

针对容器云目前基于阈值的响应式伸缩策略存在

时间滞后性、难以及时响应服务资源请求的问题,提出一种基于二次移动平均法的预测式容器云伸缩方法。该方法可以对未来工作负载进行预测并依托容器轻量特点快速做出伸缩响应。通过设计两种不同的工作负载场景对改进后容器集群伸缩方法进行了测试,验证了该方法的可行性和有效性。但该方法也存在不足,即对周期性工作负载具有较好的预测效果,而对其他类型工作负载的预测效果并不理想。后续研究可以考虑结合机器学习方法来建立预测模型,提高预测模型在各种工作负载场景下的预测准确性。

参考文献:

- [1] 张琦. 基于 Docker 的 CaaS 管理平台架构研究与设计[J]. 计算机应用与软件, 2018, 35(11): 33-41.
- [2] BERNSTEIN D. Containers and cloud: from LXC to docker to kubernetes[J]. IEEE Cloud Computing, 2014, 1(3): 81-84.
- [3] 浙江大学 SEL 实验室. Docker 容器与容器云[M]. 北京: 人民邮电出版社, 2016: 228-229.
- [4] KUKADE P P, KALE G. Survey of load balancing and scaling approaches in cloud[J]. International Journal of Emerging Trends and Technology in Computer Science, 2015, 4(1): 189-192.
- [5] LORIDO-BOTRAN T, MIGUEL-ALONSO J, LOZANO J A. A review of auto-scaling techniques for elastic applications in cloud environments[J]. Journal of Grid Computing, 2014, 12(4): 559-592.
- [6] GILLEN R E, PATTON R M, POTOK T E, et al. Cloud computing method for dynamically scaling a process across physical machine boundaries: U. S. , 8825710[P]. 2014-09-02.
- [7] VAQUERO L M, RODERO-MERINO L, BUYYA R. Dynamically scaling applications in the cloud[J]. ACM SIGCOMM Computer Communication Review, 2011, 41(1): 45-52.
- [8] 树岸, 彭鑫, 赵文耘. 基于容器技术的云计算资源自适应管理方法[J]. 计算机科学, 2017, 44(7): 120-127.
- [9] 魏锐, 刘康明. 云计算环境下多目标约束的虚拟资源动态调度[J]. 青岛科技大学学报: 自然科学版, 2014, 35(2): 210-214.
- [10] 苗立尧. 基于 Docker 容器的混合式集群伸缩方法研究[D]. 西安: 西安邮电大学, 2016.
- [11] 杨忠. 面向 Docker 容器的动态负载集群伸缩研究[J]. 舰船电子工程, 2018, 38(8): 109-115.
- [12] 刘思尧, 夏绪卫, 华荣锦. 基于 Docker 容器的云平台集群伸缩算法研究[J]. 科技通报, 2018, 34(7): 150-153.
- [13] 刘梅, 高岑, 田月, 等. 基于 Docker Swarm 集群的调度策略优化算法[J]. 计算机系统应用, 2018, 27(9): 199-204.